CHAPTER 7

Programming HTML Client Applications

Until very recently, the information contained in HTML pages was relegated to the domain of the Web browser, while standard applications concentrated on local or network data. Today, that's no longer true. Desktop applications can now work with more kinds of data than ever before, including content normally found on the Internet. You can actually create a very customized Web browser designed to look for specific kinds of Web content, or add a Web-browsing capability to a standard application that allows the user to cruise the online help desk.

The ability to combine various kinds of data into one application is a very powerful capability, but one that can also lead the programmer into trouble. Vendors have had a tendency, in recent years, to add ever more bells and whistles to applications until the application is so bloated that it actually loses functionality. The user can no longer cope with the diverse offerings of features, so many useful features remain unused. The same thing can happen with custom software written by a consultant or local company programmer — the application can become so complex and cumbersome as to lose all functionality from the user's perspective.

This chapter is designed to help you learn two essentials. First, we'll look at what an HTML-based application is all about and what it can do for you. Second, we'll look at things from the user perspective. Sure, adding HTML features to an application is a great way to integrate the Web and your local company's custom application, but you need to understand how to add that support without making the application unusable.

In This Chapter

Determining which type of HTML application to create

Learning how HTML applications differ from other application types

Working with both server and client applications

Understanding the new Internet controls

Creating an HTML client application

Adding a second view to an application

The first question we'll answer is what an HTML-based application is. We'll look at the basic application types and what kinds of situations you can use them in. Understanding how to best use an application type is at least part of the battle of implementing HTML features effectively.

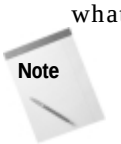
In the next section, we'll look at the Dynamic Hypertext Markup Language (DHTML). This language is a newer form of the HTML that you may be familiar with today. Because Visual C++ 6.0 supports DHTML, it's important to know exactly what that means to you, as a developer, in the way of advanced features.

Once you understand how applications and the underlying DHTML technology work together, it's time to look at the question of server versus client applications. Where do you want to implement this new technology? Sometimes the answer is obvious, but there are situations in which you may have to consider the advantages that both environments offer before making a final decision.

Microsoft has also introduced a wealth of new Internet controls. These controls can greatly ease the complexity of the coding environment for the programmer. In the fourth section of this chapter, we will discuss these new controls and how they can make you're next programming assignment a lot easier. We'll pay particular attention to the controls that make DHTML application programming possible. This section will prepare you for the next two sections, in which I'll show you actual programming examples. The first example is of a client application, the second, a server.

Understanding the HTML-based Application Types

There really is only one type of HTML-based application from an MFC AppWizard perspective — a client application that relies on the CHtmlView class to produce the view rather than something more familiar like the CEditView class. A view defines how the data gets displayed on screen rather than the data itself. An HTML-based application is actually a custom Web browser that allows you to display DHTML content in a specific way. In addition, creating an HTML-based application allows you to completely control the session — you decide where the user will go and what the user will see.



If you had used the CEditView class in place of the CHtmlView class, what you would see is the source code for the Web page rather than the Web page itself.

An application can support more than one view type, so it's possible to create a single application that displays a help desk using one view and standard data using another. This ability to display more than one view allows you to create a database application that uses online help instead of the more standard HLP file. In addition, you can add DHTML support to both MDI and SDI applications, enabling you to display the DHTML view alongside standard views in a number of ways. (Unfortunately, it takes a lot of specialized programming to add DTHML support to a dialog-based application and there really isn't a good reason to do so.)

Even though an HTML-based application is really a specialized browser in disguise, you don't need to limit the functionality of this new form of browser. Quite the contrary. You can actually create applications that are more suited to your company's needs in particular rather than have to work around the limitations of current browser technology.

The first decision you need to make is where the processing will take place, which means that you need to spend some time thinking about the content of the Web pages that you want to design and the kind of information they'll contain. DHTML allows you to execute commands on the local machine (client) or the server as needed, though, in most cases, you'll perform as much processing as possible locally to reduce the load on the server. It's the differentiation of server versus client processing that determines the kind of application you create.

There are situations when you must perform server-side processing. For example, if you need to display the contents of a database, then you'll need to write a scriptlet (JavaScript or VBScript code on your Web page that performs a certain task) that gets the data for you from the server. Although you could provide a scriptlet that would process the retrieved data locally, the initial act of getting the information would take place on the server.

We'll spend most of our time discussing the client-side processing type of application in this chapter because that's the kind of application most of you will create. Using an HTML-based application with client-side processing has many advantages as listed below:

Server Load Reduction Reducing the load on the server whenever possible is one way to ensure that you'll have the processing power needed to perform server tasks that the user requires. For example, you'll normally need to ask the server to find a particular document based on some type of search criteria. However, finding the total for the items on an order form can be done at the client rather than the server.

Speed of Processing	Client-side processing enables your application to respond faster to user requests. Calculation results will appear almost instantaneously when using client-side processing.
Reduced Network Traffic	Most network administrators jealously guard every bit of network bandwidth they can get. The reason is simple. If a Web site runs out of bandwidth, user complaints are sure to follow, and your business will grind to a halt. Client-side processing reduces the number of requests made to the server, which, in turn, reduces the amount of network bandwidth required to service a particular client.
Customization	A local process can perform tasks that the server can't. The most common task that a local process can perform is customizing a Web page for a specific user. Little things like adding the user's name can make a Web page seem more personal and friendly.
Lower Training and Support Costs	An HTML-based application can present information in a format that the user is already familiar with — the Web page. By using a standard format, you can reduce the learning curve for the user and the number of questions they have about using the application.
Special Features	You can use this format with a standard application to do things that would normally require a lot of programming. For example, you could place links right on a form to an online dictionary that explains the term in question. (Sure, you can use the Shift+F1 key to display context-sensitive help for a particular field of a form on a standard application, but links can go beyond the field level to the individual word level if necessary.)

Once you've determined where the processing will take place, you'll need to decide on a level of functionality. There are actually two levels to consider: local application and Web page.

Local application functionality determines how much freedom the user has in exploring the Web site and how much work the application does for the user. For example, you could create hard-coded links that the user could access using buttons on the toolbar. This level of functionality would restrict user movement

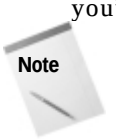
and could help ease security problems. An address bar gives the user a lot more freedom, but also means that the user could visit noncompany Web locations given the right set of circumstances. Adding a lot of user aids means that you'll have less of a learning curve for new or inexperienced users, but this same functionality means that experienced users might spend too much time worrying how to use the application and not enough time actually using the Web site.

The Web-page level of functionality determines things like the program's responsiveness as well as the features the user will see. In many cases, adding a new feature will directly impact responsiveness in some way. Obviously, if all of your users have a T1 connection, then writing large Web pages that contain a lot of features isn't much of a problem. On the other hand, if you have to worry about dial-up connections, then program responsiveness becomes a bigger issue.

You'll want to think about many other issues in regard to Web-site creation that aren't covered in this chapter — there are whole books devoted to this subject. You need to make sure that you understand both the local application and Web-page construction issues before you attempt to create an HTML-based application. This chapter will concentrate on the local application portion of the picture, though we will look at some Web-page issues as well as some very simple Web-page code.

Why DHTML?

DHTML, along with a DHTML-compatible browser like Internet Explorer 4.0 (and above), allows you to perform event-driven scripting and programming with pages that used to contain only static content. This means that a user could click a button and expect an immediate local result from your Web page. You can also create event handlers that do something outside the user's anticipated response. For example, you can create mouseover events that aren't necessarily something the user is anticipating, but that assist the user in working with the application. Event-driven programming also enables you to add timed events and other nonstatic elements like animated displays. In sum, DHTML brings Web pages closer to what you'd expect at the desktop, though not quite.



In this chapter, we use Internet Explorer exclusively as a Web browser. This is because Internet Explorer components are tied to the Internet-specific controls that we'll explore. In addition, Internet Explorer comes as part of Windows NT 2000 and Visual C++ 6.0, so I know that all of you have it available to you. None of the example code was tested with Netscape Communicator installed on the test machine, so the sample code is unlikely to work if Netscape Communicator is your only browser.

At this point, some of you may be tempted to say, "Big deal." Previous versions of both Internet Explorer and Netscape products like Communicator offered the

ability to perform some level of event-driven programming. The big difference with DHTML is that the entire environment is now open as a programming environment. No longer are you restricted to a small subset of HTML elements. (These older products offered only the opportunity to work with anchors, image maps, form elements, applets, and objects.) I will discuss DHTML in greater detail in the following sections.

DHTML Uses

One of the first major uses of DHTML is for thin clients where the ability to run traditional desktop applications is severely limited. DHTML provides the means to create Web pages that look very much like those created with a desktop application. This is because DHTML provides the exact placement and object-handling capabilities of its desktop counterpart in many ways, through OLE automation.

Using DHTML is a plus for the programmer as well because of its declarative orientation. Many of the lines of code that you used to write to resize and move components as the user resizes the application container are no longer necessary. These needs are taken care of automatically, just as they would be in any browser. In addition, the declarative language makes it easier to describe objects like command buttons.

Obviously, there's no free lunch. Using DHTML means that you are limited to using a high-end browser or OLE automation server counterpart. In addition, since the language isn't compiled, you pay a performance penalty that may leave the user waiting extended periods of time. In other words, though DHTML does provide a very flexible environment for HTML client applications, you need to use it with care.

All objects must have an ID assigned to them to make full use of the capabilities that DHTML provides. In addition, some environments require you to assign a unique ID to each object. Using a unique ID reduces ambiguity and enhances the readability of your code, so it pays to use a unique ID even if the environment you're using would allow two objects in different sections of the document to use the same ID.

DHTML Formatting Features

One of the major reasons to use DHTML is the formatting features that it provides. Sure, you won't get exactly the same kind of positioning that you would with a desktop application, but you do get a lot more flexibility than you would with standard HTML. Although this chapter won't provide a complete list of every potential DHTML formatting feature, I do provide you with an overview of what you can expect. Table 7-1 contains a list of DHTML formatting features and their descriptions.

There are some techniques you can use to reduce errors on your Web page. Two of the best techniques are Cascading Style Sheets (CSS) and relative URLs. A relative URL refers to a resource location relative to the current page location on the Web-server drive. In other words, relative links are less likely to break because they don't rely on specific locations. Style sheets ensure that your Web site has a consistent look and that it will work with all of the browsers that you've designed the style sheet to work with. In addition, style sheets enable you to format a document at the uppermost level, rather than at the individual object level. Be aware, however, that some object properties won't cascade from the parent to the child level. In this case, you need to format individual objects to ensure that the object appears properly onscreen.

Table 7-1
DHTML Formatting Feature Summary

Feature	***Description***
Automatic Resizing and Layout	DHTML supports two types of display elements: inline and block. An inline element can wrap itself around a block element. You can position a block element using relative, absolute, or percentage-based positioning. In addition, block and inline elements can be nested arbitrarily, making it easy to create complex forms.
Bottomless Forms	All forms you create with DHTML are naturally bottomless, which means repeating regions are easier to manage. In addition, DHTML automatically manages scrollbars for you, something you normally need to code as part of the application.
Graphics Types	Like HTML, DHTML supports both standard Internet graphic formats: GIF and JPEG. Besides these standard formats, DHTML also supports MPG and PNG graphics formats.
Measurement Units	Four standard measurement units are supported by DHTML: inches, centimeters, device-dependant pixels, and points (1/72 inch). Combining these absolute measurement units with the percentage units normally associated with HTML enables you to create dynamically resizable pages that contain some elements in absolute positions.
Multimedia Effects	Animation and sequencing both come as part of the DHTML package. You also have the option of attaching multimedia effects to a region of the page and applying filters as needed to achieve a desired effect.

Continued

Table 7-1 *(continued)*

Feature	*Description*
Style Sheets	Most word processing applications employ the idea of a style to help the user create complex page layouts without having to format every element of that page. Applying a style to the element is enough to format it as needed for presentation purposes. The DHTML <STYLE> tag allows you to apply this same technique to your Web pages, which greatly reduces the work required to format the presentation. You can apply styles directly or by name. In addition, DHTML allows you to change a style at runtime.
Tabular Layout	Tables have been part of HTML for quite some time. However, an HTML table is relatively static and you're limited to the formatting capabilities that HTML provides. DHTML enhances the table by allowing you to use the rich formatting capability that it provides. In addition, you can dynamically resize DHTML tables, which means that the user can reorient the table as needed for best viewing.
Typographic Support	DHTML block elements support the notion of borders, margins, and padding—the same types of elements you get with documents created using a desktop application. In addition, you can control baseline-relative positioning, leading, letter-spacing, colors, fonts, indentation, strikethrough, and the bulleting of the text.

There's a limitation on what kinds of elements you can absolutely position in a DHTML document. Only the APPLET, BUTTON, DIV, EMBED, HR, IFRAME, IMG, INPUT, MARQUEE, OBJECT, SELECT, SPAN, TABLE, TEXTAREA, and FIELDSET elements can be absolutely positioned on the page. You cannot apply absolute positioning to design-time controls (an ActiveX control that generates HTML text and script). One of the nicer aspects of absolute positioning is that you can control the X, Y, and Z-axis. The Z-axis allows you to bring an element to the front and send it to the back, which allows you a great deal of flexibility in how the element gets displayed with respect to other elements on the page.

Preventing OnLoad( ) Error Messages

Most Web pages load *asynchronously,* which means that elements that load quickly appear onscreen first. In some situations, you may find that your OnLoad( ) code executes before all of the screen elements are in place. If this happens, the user will see an error message, not the beautifully formatted page that you had anticipated. You can prevent this from happening in two ways.

You can set AsyncLoad to false, which prevents any element of the page from getting displayed before all of the elements are downloaded. The advantage of this method is that it reduces the complexity of your code, and you can always be sure that all elements required to execute the code are present. The disadvantage of this method is that users may have to wait a considerable amount of time to download the page before they see anything, making it quite likely they will click the Stop button before the page is downloaded.

The second technique is to check for the presence of an element before you use it. If you use this method, the user will see the Web page load as normal, but the complexity of your code will greatly increase as the number of elements required for the OnLoad( ) method increase.

Understanding the Event Object

Internet Explorer Versions 4.0 and above also include the idea of *event bubbling.* In the past, if an HTML event occurred and there was no handler for it, the event simply disappeared. With event bubbling, the event "bubbles" up to the parent object to look for an event handler. If there's no event handler available at the parent level, the event continues to bubble up until it either finds an event handler or runs out of levels to check, which is the document level. In short, event bubbling enables you to write a default event handler that works with most cases of an event, leaving just the special cases for special event handlers. This technology isn't new, most GUI operating systems, including Windows, have had it for quite some time. The technology is new to HTML.

Working with events means working with event objects. The event object describes what event occurred on the Web page and allows your script to do something based on what the event object contains. DHTML works with a standard set of properties for all event objects, even if the property doesn't make sense for the event object in question. For example, there is an Alt property that makes sense only when working with keypress events, but it's available in all event objects. It's up to the programmer to know when to use a specific property. Table 7-2 shows a complete list of event-object properties that are at your disposal when working with an event object.

Table 7-2
Standard Event-Object Properties

Property	*Description*
AltKey	Set to TRUE when the user presses the Alt key; always returns FALSE otherwise.
Button	Tells which mouse key was pressed. A value of 1 is the left button, 2 is the right button, and 4 is the center button (on three button mice).
CancelBubble	Allows you to stop an event from bubbling up through the object hierarchy. You can then perform custom event processing or no processing at all. In most cases, you will definitely want to use this property after custom processing to prevent the event from traveling up the hierarchy and getting processed twice (once with the custom function and again with the default function).
ClientX	The X position of the mouse within the client window.
ClientY	The Y position of the mouse within the client window.
CtrlKey	Set to TRUE when the user presses the Ctrl key; always returns FALSE otherwise.
FromElement	Returns the name of the element that the mouse was on during an onmouseover or an onmouseout event.
KeyCode	Returns the Unicode value of the keypress.
OffsetX	The X position of the mouse relative to the current container.
OffsetY	The Y position of the mouse relative to the current container.
Reason	Defines the reason of the completion of a data transfer when working with a data-source object.
ReturnValue	Allows you to provide a return value for the event.
ScreenX	The X position relative to the screen size (the meaning of the term *position* is defined by the object that generated the event).
ScreenY	The Y position relative to the screen size (the meaning of the term *position* is defined by the object that generated the event).
ShiftKey	Set to TRUE when the user presses the Shift key; always returns FALSE otherwise.
ScrElement	The name of source element that fired the event.
ScrFilter	The name of the source filter that fired an onfilterchange event.
ToElement	Returns the name of the element that the mouse moved to during an onmouseover or an onmouseout event.
Type	The event name.

Property	*Description*
X	The element's X position, relative to the parent hierarchy. It is determined using cascading style sheet (CSS) positioning.
Y	The element's Y position, relative to the parent hierarchy. It is determined using cascading style sheet (CSS) positioning.

Microsoft provides a lot of online resources that will help you learn the current state of HTML, DHTML, and CSS. This includes toolkits along with the latest standards information (and the Microsoft view of those standards). The main Web site for this information is at `http://www.microsoft.com/workshop/author/default.asp`. A DHMTL Editing Component SDK is also available that will make using the full capabilities of DHTML in a variety of environments much easier. In essence, the SDK makes it possible for you to add editing capabilities to your Web site (or other HTML-based application). These added DHTML capabilities rely on Component Object Model (COM) to do their work. You'll be able to add standard editor features like table and text formatting to your Web site. In addition, you can add advanced features like Undo to the Web site to make it easier for the user to interact with the edited text. You'll find information about the DHTML Editing Component SDK at `http://www.microsoft.com/workshop/author/dhtml/edit/default.asp`. In addition, Microsoft provides a newsgroup, microsoft.public.inetsdk.programming.dhtml_editing, where you can get DHTML Editing Component questions answered. Another good place to get information about the DHTML Editing Component SDK is your Microsoft Developer Network (MSDN) subscription.

Working with the New Internet Controls

Visual C++ includes several new controls that make it easier to work with the Internet. Most of these new controls are grouped together as the Internet Explorer Common Controls. You'll access these controls through the CComboBoxEx, CDateTimeCtrl, CIPAddressCtrl, and CMonthCalCtrl. In addition to these four classes, you'll find a new version of the CHtmlView class that encapsulates the functionality of the Web Browser control. Table 7-3 describes each of these new classes in detail.

Table 7-3
New Internet Control Class Descriptions

Class	*Control*	*Description*
CComboBoxEx	Combo Box	Allows you to use image lists with an extended combo box control. Otherwise, this new control has all the features of a standard combo box.
CDateTimeCtrl	Date/Time Picker	This control has two new interfaces: extended edit box or a simple calendar. In the first case, the control allows the user to choose a specific time or date. The simple calendar interface will allow only the user to choose a date.
ChtmlView	Web Browser	This class relies on the Web Browser control, rather than providing a wrapper for it. Two functions are performed by this class. First, it allows the application to parse DHTML script. Second, it allows you to create a general-purpose browser that can be augmented through the use of other support controls.
CIPAddressCtrl	IP Address	Creates an edit box that enables the user to edit or enter an IP address.
CMonthCalCtrl	Calendar	Displays a one-month calendar that allows the user to choose a date using a standard reference. This control, unlike the Date/Time Picker, does not allow you to manipulate the time. However, the format of the calendar is easier to use than an edit box for some users.

Don't think that the Visual C++ common and dialog editor controls found in previous versions have remained the same. Most of the controls have been updated to provide a new appearance or functionality. The appearance additions are designed to match those found in Internet Explorer 4.0 and above. For example, you now have access to the flat toolbars and image lists used by Internet Explorer 4.0.

Of particular importance is a change to the CInternetSession class. You can now work with cookies using this class. There are methods for setting and getting the cookie. The class will also allow you to determine the size of the cookie so that you can create an appropriate buffer. Cookies are especially important for storing individual user settings so that you can create a site tailored to that individual's needs.

Writing a Client Application

You can use the HTML client class, CHtmlView (provided with Visual C++), in a lot of different ways. Two of the more common ways to use this class include creating a custom browser that limits employee activity on the Internet and replacing the common help (HLP) file found in most desktop applications.

Let's discuss the first common use, limiting employee interaction with the Internet. It's probably safe to say that most large companies don't want their employees perusing the game Web sites on company time. You want to ensure that employees with Internet access are actually pursuing company goals. One way to ensure this is to create a custom browser that allows access to only a few Internet (or intranet) sites.

Even though this approach reeks of big brother looking over the employee's shoulder, there are advantages for the employee as well. One of the biggest advantages is reduced application complexity. You can create a custom application that provides very specific guidance on locating and using company resources on the Internet. In addition, you may not even need to tell the employee that he or she is using the Internet in some situations since the custom application could look like any other application on the employee's desktop. Another advantage is that you can add the HTML view to the application in several ways — enhancing application functionality. For example, you can add the view in such a way that the employee can read HTML files on his or her hard drive without exiting the custom application.

Now that we've gotten the restrictive example out of the way, let's discuss the help file replacement application. One of the biggest problems with help files is that they are inherently inaccurate. The help file normally gets written while the application is getting written, which means that last-minute application changes may not even appear in the help file. In addition, even the best writer will miss details or get some facts wrong. A help file that is stored on the user's local hard drive is just too hard to update, in most cases, to make "reprint corrections" that fix these errors.

Another problem with help files is that they consume hard drive space. The bigger and more complex the application, the more space the help file will require. Though this isn't a big problem on desktop machines with gigabytes of hard drive space, it is a problem on laptops — especially older laptops with limited expansion potential. Some users will leave the help file behind to make space for data files or other applications.

Help files are also inflexible. Your support staff will gain experience with the application as users ask questions about it. Unfortunately, a help file won't allow the support staff to add things like a FAQ (frequently asked questions) sheet or to reorganize the help information so that commonly required information appears first.

XML—A Future Addition?

DHTML isn't the latest technology to hit the market, even if it is the latest technology that is directly supported by the Visual C++ Web controls. Extensible Markup Language (XML) is the latest technology under consideration by both companies like Microsoft, and by standards committees like the World Wide Web Consortium (W3C). One of the best places to find out more general information about XML and access to additional links is the XML References, Resources, and Demos Web site at `http://www.cen.com/ng-html/xml/xml-refs.html`.

XML is described as a method for defining, sharing, and verifying documents on the Internet (though it can obviously be used on intranets as well). The level of document description is far better than HTML, which is one of the main reasons for creating this new specification. XML is a subset of the same Standardized Generalized Markup Language (SGML) that has been used for other page-description technologies.

The XML specification uses words like vendor independence and user extensibility as part of the description of this advanced method for describing Web-page content. In fact, that's a large part of the power of XML—the ability to extend the language to meet specific application needs. You can find out more about the metamorphosis of XML by looking at the W3C specification information at `http://www.w3.org/XML/`.

The ability to extend XML using unique data markup tags by anyone was a source of concern for many industry officials. They feared that chaos would result as thousands of undocumented tags hit the market. As a result, there was an effort to create standardized tags, which will definitely affect what you see in the way of documented XML. However, a recent addition to the picture of XML Namespaces will still allow an individual to create unique data markup tags, provided they also include a reference to the XML Namespace at the beginning of the document. An outcome of this new feature is that you'll be able to create very complex and highly formatted documents from multiple sources that use unique tags to create a specific appearance. In other words, your Web-site content will no longer be subject to the quirks of the HTML specification.

The centerpiece of XML is the XML engine, which is used to interpret the various tags and other XML entities. The XML engine is where a vendor can add functionality to the basic XML outlined by the W3C specification. Obviously, the idea of using an engine of some sort to interpret the human readable form of a language is not new—BASIC began as an interpreted language, as did many others. What's important to know is that the XML engine is what gives this language the features that it enjoys when used on a particular platform. You can find a wealth of demos that show you what XML is all about at the Microsoft XML Web site at `http://www.microsoft.com/workshop/c-frame.htm#/xml/default.asp`.

Looking at the variety of demos that Microsoft provides is enlightening because you get to see XML in action. Gone are the kludges required to make pages written with HTML look good. In addition, you'll find that a simple change of stylesheet can make a big difference in the appearance of the page when using the same data. Keeping the data separate from the stylesheet means that you have to create either element only once.

At first, it may appear that XML will eventually replace DHTML. However, there are many ways in which DHTML and XML are meant to complement each other—at least, if you're using the technologies from the Microsoft perspective. One of the most important technology overlaps comes in the form of scriptlets. A scriptlet enables you to define event handlers and a wealth of other programming constructs.

Both DHTML and XML allow you to create scriptlets that can expose properties and methods, and fire events. However, the two technologies have some very important differences. The most important difference from a usage perspective is that DHTML is better suited to creating objects with a user interface. XML, on the other hand, is an automation object without a user interface and object model to rely on.

The Microsoft version of XML relies on the component object model (COM). This is the same technology used to create components for Microsoft programming languages and the Windows operating system environment. You'll find COM in every area of Microsoft products since everything relies on objects created using COM.

Using the Microsoft vision as a point of comparison, DHTML and XML have some very basic differences from a programmer's perspective. The programming interface for a DHTML scriptlet is the JavaScript function; XML uses a type library. Creating a DHTML object and finding out about it means calling the public_description JavaScript function. You perform the same task for XML using the DllGetClassObject( ) function, then use the COM automation interface. Events are fired using DHTML's window and external objects, whereas XML uses the COM connection pointer interface.

It may seem unlikely at first that you can create a COM object using scriptlets created with XML. Microsoft's secret weapon is SCROBJ.DLL. This is the DLL that clients interact with when attempting to access an XML scriptlet that includes the necessary COM interfaces. In other words, SCROBJ.DLL is the in-process server for XML. There are other COM-specific attributes for XML as well, like object entries in the registry and direct instantiation.

Don't get the idea that XML allows you to use COM fully, nor can you implement any portion of COM fully using XML. For example, when working with an interface handler, you can implement only defined interfaces—you can't create a new interface of your own. In addition, you can't create your own handlers; the XML scriptlet must rely on SCROBJ.DLL. (The interface handler sits between the client and the XML scriptlet, enabling them to communicate as needed using standard interfaces and connection points.)

As you can see, XML is a great new technology. Hopefully, Microsoft will begin incorporating this functionality into Windows and as part of the Win32 API as you read this. Unfortunately, you'll probably need to wait for the next version of Visual C++ before this new functionality will appear in an easy-to-use form for the Visual C++ programmer.

As you can see, creating an online help desk for your application is important for ensuring the user gets the best possible information all of the time. Online help desks are:

- ✦ Easy to update.
- ✦ Allow intermediate changes as needed to fix even small errors.
- ✦ Require no hard drive space on the user's machine.
- ✦ Benefit from the support staff's increased knowledge about the application.

This example will actually show two different techniques for using the CHtmlView class. First, we'll add the capability of viewing the HTML files directly by adding a second view to the application. Second, we'll add a Help Desk menu option that will take the user directly to an on-site source of information.

CD-ROM Location: Chapt07\HTMLView
Program Name: HTMLView.EXE
Source Code Modules in Text: HelpDoc.H, HelpDoc.CPP, HelpView.H, HelpView.CPP, HTMLView.RC, HTLMView.CPP, MainFrm.H, MainFrm.CPP

Creating the Program Shell

The first thing we'll need to do is create a program shell. In this case, we'll create an MDI application to make it easier to use multiple views at the same time. When writing a production application, you need to decide whether you need an SDI or MDI to achieve the goals of the application as a whole, then add the CHtmlView class view within those constraints. The following procedure will get you started (some steps are abbreviated in this procedure — see Chapter 4 for the most comprehensive application creation procedures).

Try This

1. Create a new MFC AppWizard (exe) project. The example uses a name of HTMLView, though you could use any name you wanted. You'll see the MFC AppWizard - Step 1 dialog.
2. Click Next three times. You'll see the MFC AppWizard - Step 4 of 6 dialog.
3. Click Advanced. You'll see the Advanced Options dialog shown in Figure 7-1. This is the dialog that allows you to define the Main View document template options for your application. A document template, as we'll see a bit later, includes settings such as the file extension associated with the view.

Most commercial applications still use either three- or four-letter file extensions. You can ensure that your custom application won't interfere with any current commercial application by using five or more letters for the file extension of your application data. Since the user won't normally need to worry about the file extension anyway, the longer file extension shouldn't cause any problems. Obviously, you'll still need to use three-letter file extensions if your application will interact with older machines that only support three-letter file extensions.

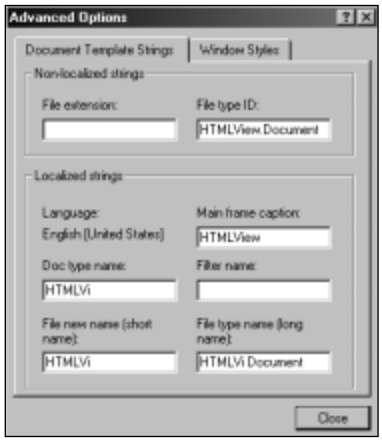


Figure 7-1: The Document Template Strings tab of the Advanced Options dialog allows you to define various view options such as the file extension.

4. Type **txt** in the File extension field. Our text editor will work with text files normally.
5. Open the Registry Editor and look for the .txt entry under HKEY_CLASSES_ROOT. Your entry should look similar to the one shown in Figure 7-2. Notice that the default entry is txtfile — that's the file type ID value.

Figure 7-2: When working with an existing document type, always check the registry for the correct file-type ID value.

6. Type **txtfile** in the File type ID field. Now we need to define some localized strings that get used by the application during the opening and display process.
7. Type **HTMLView Text Editor** in the Main frame caption field, **Text Files (*.txt)** in the Filter name field, **Text File** in the File new name (short name) field, and **HTMLView Text Document** in the File type name (long name) field. Click OK to complete the document template string definition process.
8. Click Next twice. You'll see the MFC AppWizard - Step 6 of 6 dialog shown in Figure 7-3. Notice that the Based class drop-down listbox contains a number of class entries. You can choose any of these entries as the base class for the view used for your application. The CView class is the most generic of these entries and is really designed only to allow you to create completely new data views, not views that could be derived from existing views.

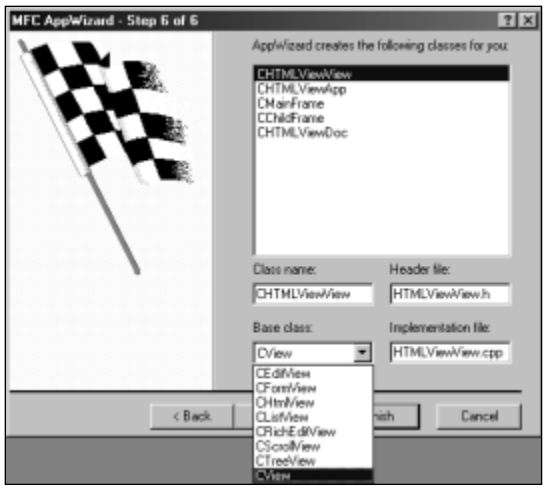


Figure 7-3: The Base class field allows you to define what view class will be used to display the data managed by your application.

9. Choose CEditView in the Base class field. This view provides very simple, nonformatted text-editing capabilities. The CRichEditView class is a better option if you want to create formatted text with your application. You'll also notice a CFormView option that is used to create data-entry forms and the CHtmlView class that we'll use later for viewing Web sites.

10. Click Finish. You'll see the New Project Information dialog shown in Figure 7-4.

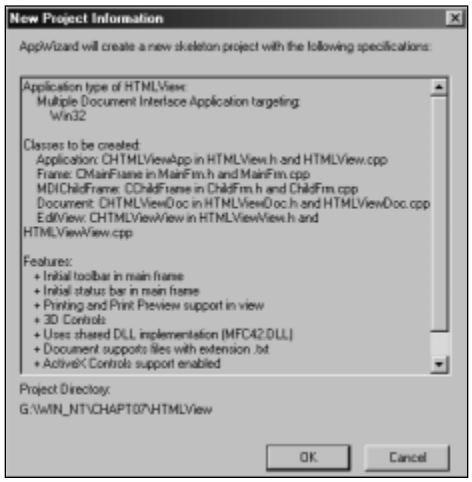

New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of HTMLView:
Multiple Document Interface Application targeting:
Win32
Classes to be created:
Application: CHTMLViewApp in HTMLView.h and HTMLView.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
MDIChildFrame: CChildFrame in ChildFrm.h and ChildFrm.cpp
Document: CHTMLViewDoc in HTMLViewDoc.h and HTMLViewDoc.cpp
EditView: CHTMLViewView in HTMLViewView.h and HTMLViewView.cpp
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Printing and Print Preview support in view
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ Document supports files with extension .txt
+ ActiveX Controls support enabled
Project Directory:
G:\WIN_NT\CHAPT07\HTMLView
OK
Cancel

Figure 7-4: The New Project Information dialog shows which class we'll use for the application view, along with other application statistics.

11. Click OK. Visual C++ will create the application.

Adding a View

If you compiled the application, at this point, you can view and edit text files. However, one of the primary goals of this example is to be able to view HTML files, which means adding another view. This view will rely on the CHtmlView class mentioned earlier.

The several major steps in the view creation process are:

1. Add a new view class (in this case, CHtmlView). The view handles the appearance of the document in the client window.
2. Add a new document class, which is normally based on the CDocument class. The new document class will handle details like creating a new document when the user requests one.
3. Define the required resources to use the view, including the menu, icon, and template string.
4. Add the new document template to the application file.
5. Create any required message handlers for the view.

Obviously, this is a very simple overview of a more complex process. In the following sections, I show you how to perform all five steps.

Adding the View and Document Classes

Before we can do anything, there has to be a view and document class to work with. That's what we'll take care of in this section, adding the required classes to our application so that the user will be able to open more than one document type. The following procedure will show you how.

Try This

1. Right-click HTMLView classes in the ClassView tab of the Workspace window, then choose New Class from the context menu. You'll see the New Class dialog shown in Figure 7-5. This is where you define the new class name, any associated dialog, and the base class that the new class is derived from. Notice that you have a choice of MFC Class, Generic Class, and Form Class in the Class type field.
2. Type **CHelpView** in the Name field and chose CHtmlView as the base class. We won't need to add a dialog, in this case, because the CHtmlView class doesn't require one.
3. Click OK to add the class.
4. Perform steps 1 through 3 again. However, this time give the class a name of CHelpDoc and base it on the CDocument class.

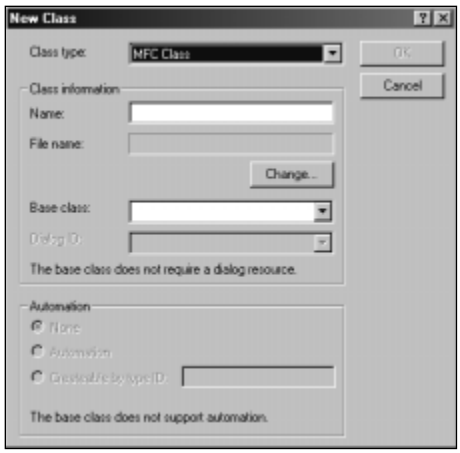


Figure 7-5: The New Class dialog allows you to define new classes for your application.

Defining the Required Resources

The ResourceView tab of the Workspace window contains a list of every resource used by the current project. Every view can use resources to identify various view features like the icon associated with the view. These resources include template string, menu, icon, and toolbar. You don't necessarily have to use all of the resources for any view, but all of the resources associated with the view do have to have the same ID value. This ID value is provided as part of the CMultiDocTemplate( ) function call (used to create the document template) that we'll discuss later in the chapter.

Adding a Document Template String

Let's begin by talking about the template string resource. This is the one resource you should always include for any view since it defines what will appear in places like the File Open dialog. Seven substrings are associated with the template string resource as listed here:

- **Window Title (SDI Applications Only):** Changes the value shown in the application's title bar when the user creates a new document.
- **Document Name:** This is the default document name. The application will combine this name with a sheet number. For example, if you supply a value of MyDoc, then the first document the user creates will be MyDoc1. Untitled is the default value for this string.
- **New File Document Name:** If your application supports multiple views that can create new files, then the application will display a dialog asking the user what kind of document they want to create when using the File ➪ New

command. This string determines the document name that will appear in the New File dialog. You normally need to use a descriptive, rather than an exact name; in this case, like worksheet. If you don't supply a value for this substring, then the user won't be able to create new files with this view.

- **Filter Name:** The filter string gets displayed in several common dialogs. It's usually a description document name along with the filter extension. For example, you might supply a value of "Text Files (*.txt)" for a text file filter name. If you don't supply a value for this substring, then the user won't be able to use the File ➪ Open command to open a file of this type.
- **Filter Extension:** This is the file extension associated with the template. For example, you might supply a value of *.txt for a text file template. If you don't supply a value for this substring, then the user won't be able to use the File ➪ Open command to open a file of this type.
- **File Type ID:** Use this substring to define the registry document type. In other words, this substring is used to create the key in the HKEY_CLASSES_ROOT hive of the registry. The user will never see this substring. However, if you don't supply this substring, then the user won't be able to open files of this type using applications like Windows Explorer. (They can still open the file from within the application itself.)
- **File Type Name:** This is the name of the document as you want it stored in the registry. Some applications use this value when displaying information about the file, so a descriptive name is always a plus. Make certain that you supply this value if you supply a file-type ID. Otherwise, the user may see a blank file type identification in some applications.

Now that you have some idea of what the template string resource is used for, it's time to create one. Follow the procedure below:

Try This

1. Open the String Table resource by double-clicking the String Table entry in the String Table folder shown on the ResourceView tab of the Workspace window. You'll see a String Table window like the one shown in Figure 7-6. Notice that the IDR_HTMLVITYPE string is highlighted. This is the document template string for the text files that the example application normally works with. Notice that each of the template substrings is separated by the \n (newline) escape sequence.
2. Right-click the IDR_HTMLVITYPE entry, then choose New String from the context menu. You should see a String Properties dialog like the one shown in Figure 7-7. Notice that this dialog contains the numeric value of the ID in the ID field and that the string will appear in sequence immediately after the existing document template string.

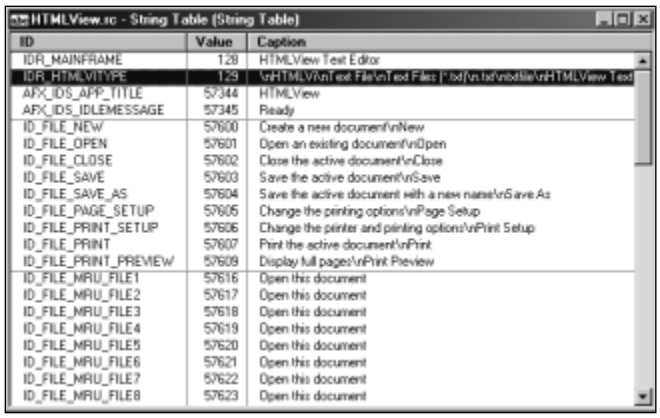

HTMLView.rc - String Table (String Table)
ID
Value
Caption
IDR_MAINFRAME 128 HTMLView Text Editor
IDR_HTMLVITYPE 129 \nHTMLV\nText File\nText Files (*.txt)\n.txt\ntxtfile\nHTMLView Text
AFX_IDS_APP_TITLE 57344 HTMLView
AFX_IDS_IDLEMESSAGE 57345 Ready
ID_FILE_NEW 57600 Create a new document\nNew
ID_FILE_OPEN 57601 Open an existing document\nOpen
ID_FILE_CLOSE 57602 Close the active document\nClose
ID_FILE_SAVE 57603 Save the active document\nSave
ID_FILE_SAVE_AS 57604 Save the active document with a new name\nSave As
ID_FILE_PAGE_SETUP 57605 Change the printing options\nPage Setup
ID_FILE_PRINT_SETUP 57606 Change the printer and printing options\nPrint Setup
ID_FILE_PRINT 57607 Print the active document\nPrint
ID_FILE_PRINT_PREVIEW 57609 Display full pages\nPrint Preview
ID_FILE_MRU_FILE1 57616 Open this document
ID_FILE_MRU_FILE2 57617 Open this document
ID_FILE_MRU_FILE3 57618 Open this document
ID_FILE_MRU_FILE4 57619 Open this document
ID_FILE_MRU_FILE5 57620 Open this document
ID_FILE_MRU_FILE6 57621 Open this document
ID_FILE_MRU_FILE7 57622 Open this document
ID_FILE_MRU_FILE8 57623 Open this document

Figure 7-6: The String Table is where string constants are stored, including the one that defines the document template string.

Figure 7-7: The String Properties dialog allows you to define the string constant associated with a particular string resource.

3. Type **IDR_HELPVIEWTYPE** in the ID field. This is the ID that we'll use for all of the resources created for the document template associated with the HTML view.
4. Type **\n\n\nHTML Files (*.htm)\n.htm\nhtmlfile\nHTML File** in the Caption field. Remember that the caption is composed of seven substrings, as mentioned earlier. In this case, we'll allow the user to open HTML files, but not create new ones. I retrieved the "html" and "HTML File" substrings from the registry using RegEdit. Since these substrings are already defined as part of the initial Windows installation, there is no need to define new ones.
5. Close the String Properties dialog. You'll see a new string appear in the String Table window with a value of 130.

Adding an Icon

The next thing we'll do is create an icon for the document template. In fact, we actually need two icons: one that is 16 (16 pixels for the small icon view and another that is 32 (32 for the large icon view. Follow the procedure below to create the icon needed for this example.

Try This

1. Right-click the Icon folder in ResourceView, then choose Insert Icon from the context menu. You'll see a new icon added to the list of icons in the folder. It should have an ID of IDI_ICON1 if this is the first new icon that you've added; it will be assumed that it is in the following steps. We need to change the ID of the icon to match the document template string created earlier.
2. Right-click the IDI_ICON1 entry, then choose Properties from the context menu. You'll see an Icon Properties dialog like the one shown in Figure 7-8. This is where we'll change the ID of the icon.

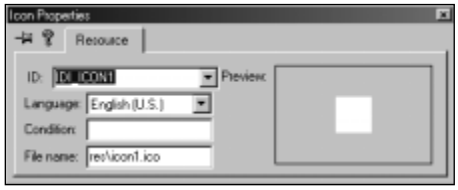


Figure 7-8: The Icon Properties dialog allows you to change the ID of the icon and provides a preview of what the icon will look like.

3. Type **IDR_HELPVIEWTYPE** in the ID field, then close the Icon Properties dialog. Notice that Visual C++ supplies a standard 32 (32-pixel icon, so we won't need to add one for this example. We do, however, need to add a 16 (16-pixel icon for the small icon listing in Windows Explorer.
4. Click New Device Image (Ins) on the Icon window. You'll see the New Icon Image dialog shown in Figure 7-9. This is where you'll choose one of the standard icon sizes that Windows uses. You can also define a custom icon size as we did in Chapter 4.

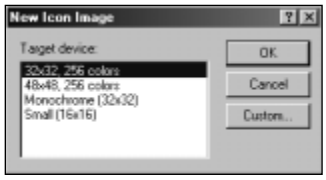


Figure 7-9: The New Icon Image dialog allows you to create new standard or custom icons of any size.

5. Highlight the Small (16(16) entry, then click OK. Visual C++ will add a second icon to the IDR_HELPVIEWTYPE icon resource. You can choose between the two icons by selecting the one you want to see in the Device drop-down listbox.
6. Draw the icons for this document template. The icon you draw here won't affect the icon for the text-file document template created as the application's default view. Figure 7-10 shows the 32 (32-pixel version of the icon I created for this example. You can create any icon you wish (or leave it blank if you don't want to see an icon in the Windows Explorer display).

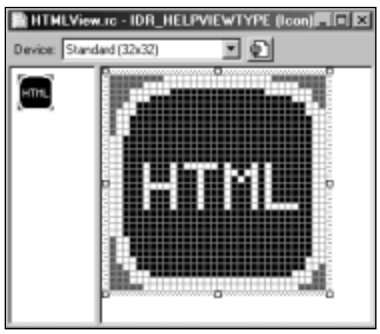


Figure 7-10: This is the icon that Windows Explorer (and any other application that displays icons) will use when working with this document type.

Adding a Menu and Toolbar

There are a lot of situations in which you can use a standard toolbar and menu for all of the views that your application supports. However, this isn't one of them. The HTML view needs movement commands for both the menu and the toolbar. At a minimum, the user will need a Back button so that they can move back to the previous page. Though it is optional, a Forward button always comes in handy. Obviously, a standard browser provides a lot of other buttons like an Address field, which would appear only on the toolbar. You could provide these additional buttons for your application, but they aren't absolutely necessary.

Two buttons that you may want to add for this kind of application that won't appear on many standard browsers are a Company Page and a Help desk button. These two buttons will help the user get more out of your custom application while also making it simpler to operate.

Let's begin with the toolbar. Obviously, you don't want to spend all of your time creating a specialized version of the toolbar supplied with the application. Yet, you'll want to be able to customize that toolbar to meet the needs of this particular view. For example, the user doesn't need a New button because they can't create new HTML documents with this application. The following procedure will show you how to create a custom view toolbar with the least amount of work.

Try This

1. Right-click the IDR_MAINFRAME toolbar within the Toolbar folder in ResourceView, then choose Insert Copy from the context menu. You'll see the Insert Resource Copy dialog shown in Figure 7-11.

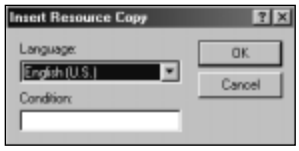


Figure 7-11: The Insert Resource Copy dialog allows you to create a new version of an existing toolbar without redrawing it.

2. Type any letter in the Condition field, then click OK. (The reason for typing a letter in the condition field is that otherwise Visual C++ won't create the copy for you.) You'll see a new toolbar added to the list of toolbars in the folder. It should have an ID of IDR_MAINFRAME with the condition you typed in parenthesis. We need to change the ID of the toolbar to match the document template string created earlier and remove the condition.
3. Right-click the IDR_MAINFRAME copy entry, then choose Properties from the context menu. You'll see the Toolbar Properties dialog shown in Figure 7-12. This is where we'll change the ID of the toolbar and remove the condition.

Figure 7-12: The Toolbar Properties dialog allows you to change several toolbar characteristics, including the ID.

4. Choose the IDR_HELPVIEWTYPE in the ID drop-down listbox (you can always choose one of the existing IDs rather than retype it). Delete the letter you typed in the Condition field. Close the Toolbar Properties dialog to make the change permanent. At this point, you have a copy of the IDR_MAINFRAME toolbar that you can customize for this view.
5. Highlight the New button, then drag it from the toolbar. This action deletes the New button.
6. Highlight the empty button next to the Help button and move it slightly to the right. Draw a left arrow on this button to indicate that the user can use it to move backward.
7. Double-click the new Back button. You'll see the Toolbar Button Properties dialog shown in Figure 7-13. This is where you define the ID and other characteristics of the button. If you were creating a button and menu combination, then the button and the menu entry would need to have the same ID to work together.

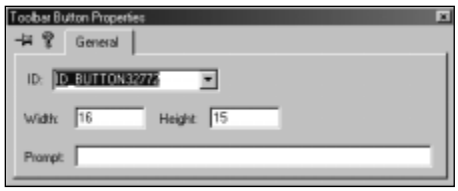


Figure 7-13: The Toolbar Button Properties dialog allows you to define the ID, prompt, and size of the toolbar button.

8. Type **ID_GO_BACK** in the ID field. This ID will be used for a variety of purposes, including the message handler we'll create later.
9. Type **Go back to the previous Web page.\nGo Back** in the Prompt field. The first substring will appear on the status bar of the application; the second as balloon help when the user passes the mouse cursor over the button.
10. Perform steps 5 through 8 again to create a Forward button. Give this button an ID of ID_GO_FORWARD, and a prompt of "Go forward to the previous Web page.\nGo Forward." At this point, your toolbar should look similar to the one shown in Figure 7-14.

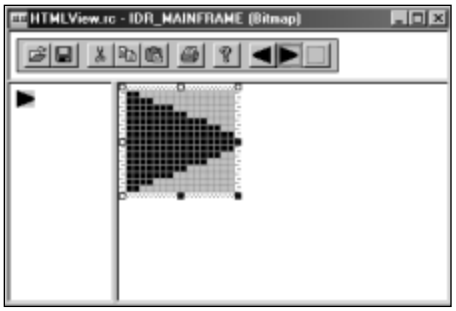


Figure 7-14: The new toolbar sports special buttons for moving around on the company's help desk intranet site.

You'll find out later that the application won't automatically load this toolbar for you; it isn't included as one of the items automatically tracked for you by the template. However, we'll add code to load the custom toolbar as needed for working with HTML documents. Fortunately, the menu that we'll add next is automatically handled by the template code.

Now that we have a custom toolbar to use, we can add a custom menu to go with it. This procedure uses the same technique as the previous one. We'll simply copy a menu and customize it to meet our needs. However, in this case, we'll copy the menu used for the text view, IDR_HTMLVITYPE, because it's more complete than the default menu, IDR_MAINFRAME.

Try This

1. Right-click the IDR_HTMLVITYPE menu within the Menu folder in ResourceView, then choose Insert Copy from the context menu. You'll see the Insert Resource Copy dialog shown in Figure 7-11.
2. Type any letter in the Condition field, then click OK. You'll see a new menu added to the list of menus in the folder. It should have an ID of IDR_HTMLVITYPE with the condition you typed in parentheses. We need to change the ID of the toolbar to match the document template string created earlier and remove the condition.

3. Right-click the IDR_HTMLVITYPE copy entry, then choose Properties from the context menu. You'll see the Menu Properties dialog shown in Figure 7-15. This is where we'll change the ID of the toolbar and remove the condition.

Figure 7-15: The Toolbar Properties dialog allows you to change several toolbar characteristics, including the ID.

4. Choose the IDR_HELPVIEWTYPE in the ID drop-down listbox. Delete the letter you typed in the Condition field. Close the Menu Properties dialog to make the change permanent. At this point, you have a copy of the IDR_HTMLVITYPE menu that you can customize for this view. In this case, all we'll do is add a Go menu with options for going forward and backward.

5. Type **&Go** in the blank menu entry after Help. Press Enter to close the Menu Properties dialog that appears.

6. Move the Go entry by dragging it between View and Window.

7. Double-click the first submenu entry after the Go menu entry. You'll see the Menu Properties dialog shown in Figure 7-16. We'll add the Go | Forward command first.

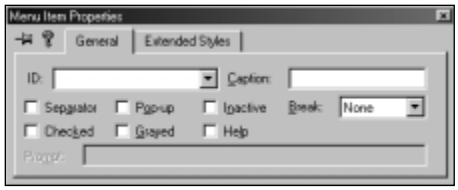


Figure 7-16: The Menu Properties dialog allows you to define the characteristics of a menu entry.

8. Choose ID_GO_FORWARD in the ID field and type **&Forward** in the Caption field. Now we're ready to add the Go | Back command.

9. Double-click the second submenu entry. Choose ID_GO_BACK in the ID field and type &Back in the Caption field. Your new menu should look like the one shown in Figure 7-17.

Figure 7-17: The Go menu will allow the user to use commands to move around on a Web page.

Adding a Document Template

Two sections of code are required to implement the document template itself. Both sections appear in the HTMLView.CPP file. The first thing you'll need to do is add two #include directives at the beginning of the file that allow the program access to the view and document class files. Listing 7-1 shows the code you'll need to add.

Listing 7-1: The #include Directives Required to Implement a Template

```
// Include the two files holding our specialty
// HTML view document and view.
#include "HelpDoc.h"
#include "HelpView.h"
```

Once you have the directives in place, you can modify the InitInstance( ) method to include the new template. Listing 7-2 shows this code in bold (the surrounding code is provided so that you know where to place the code).

Listing 7-2: Modifications Required to the InitInstance( ) Method to Add a Template

```
    CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
        IDR_HTMLVITYPE,
```

```
    RUNTIME_CLASS(CHTMLViewDoc),
    RUNTIME_CLASS(CChildFrame), // custom MDI child frame
    RUNTIME_CLASS(CHTMLViewView));
AddDocTemplate(pDocTemplate);

// Add a custom HTML view.
pDocTemplate = new CMultiDocTemplate(
    IDR_HELPVIEWTYPE,
    RUNTIME_CLASS(CHelpDoc),
    RUNTIME_CLASS(CChildFrame), // custom MDI child frame
    RUNTIME_CLASS(CHelpView));
AddDocTemplate(pDocTemplate);

// create main MDI Frame window
CMainFrame* pMainFrame = new CMainFrame;
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
    return FALSE;
m_pMainWnd = pMainFrame;
```

The template code isn't very complicated. All you need to provide is the ID of any resources that you want to use, along with the class name of the document, frame, and view classes that you want to use. The function you choose to use for creating a document template pointer containing the required information depends on the type of application you've created. Our example shows the CMultiDocTemplate( ), which is the function you'd use for an MDI application. When creating an SDI view, you'd use the CSingleDocTemplate( ) function instead. Once you have a document template pointer, you add it to the application using the AddDocTemplate( ) function.

If you compiled the application, at this point you'd be able to open two types of files: text and HTML. However, the HTML view is far from functional. It still doesn't include the code required to work with the document that the user selects from the file list or take the user to the help desk. We'll complete this part of the view in the next section.

Creating the Required View Message Handlers

We'll need to add several message handlers to make the HTML view work. Microsoft assumes nothing about the view that you want to create, which means that you'll pretty much start from scratch as you create the code required to make the view work. This has the advantage of allowing you to create a very customized view without wading through a lot of template code. On the other hand, this approach does make it difficult for the first-time programmer to figure out what to do. The approach we'll take in this section is minimalist. In other words, if you're creating an HTML view, this is about the least amount of work that you should consider doing to implement it.

In many cases, the programmer has several choices when it comes to working in unfamiliar territory. You can look through the help files that come with Visual C++ and hope they'll contain what you need, read books such as this one, or search through the source code that comes with products like MSDN. However, in this case, there's one other source of information that you should consider looking at—the text view that the MFC AppWizard created for you. This view is fully functional and contains many helpful clues you can use to create an alternate view of your own.

Displaying an HTML Page

Let's begin by making the view functional. Right now it can't display anything even if we do supply it with a place to look. Expand the CHelpView and CHTMLViewView class hierarchies in ClassView. If you compare the list of implemented methods between the two views (text and HTML), you'll find that the HelpView.CPP file is missing the GetDocument( ) method. This very special method is a required addition if you want to work with the document at all, which includes getting the name of the file that you're supposed to view after the user makes a selection in the File Open dialog.

The MFC AppWizard takes a very convoluted approach to creating this particular method and I'd suggest you follow that approach if you want the new view to work as intended in a variety of environments. The first thing you need to add is a function declaration (a required addition for any method) as is shown in bold in Listing 7-3.

Listing 7-3: Adding the GetDocument( ) Method Declaration

```
// Attributes
public:
    CHelpDoc* GetDocument();

// Operations
public:
```

There are actually two different code sections for this method. One is used for the release version of the program. The other is used for debugging purposes. Both versions perform the same work, but they appear in different places. Listing 7-4 shows the code required for the release version of the method in bold, and Listing 7-5 shows the code required for the debug version in bold. Note that you place the release version in the header file, but that the debug version goes in the CPP file. In addition, you must place the release version of the GetDocument( ) method after the class declaration as shown in the listing.

Listing 7-4: Release Version of GetDocument() Method (HelpView.H)

```
    DECLARE_MESSAGE_MAP()
};

// This version of GetDocument is used in the release
// version of the product. The debug version is found
// in HelpView.CPP.
#ifndef _DEBUG
inline CHelpDoc* CHelpView::GetDocument()
   { return (CHelpDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
```

The view class normally contains a diagnostic section where you can place the GetDocument( ) method. If you don't see this section of code, then you'll need to add the #ifdef _DEBUG and #endif //_DEBUG directives to the beginning and end of the code as shown in Listing 7-5.

Listing 7-5: Debug Version of GetDocument( ) Method (HelpView.CPP)

```
// CHelpView diagnostics

#ifdef _DEBUG
void CHelpView::AssertValid() const
{
    CHtmlView::AssertValid();
}

void CHelpView::Dump(CDumpContext& dc) const
{
    CHtmlView::Dump(dc);
}

// This is the debug version of the GetDocument() method.
// You'll find the release version in the HTMLView.H file.
CHelpDoc* CHelpView::GetDocument()
```

Continued

Listing 7-5: *(continued)*

```
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CHelpDoc)));
    return (CHelpDoc*)m_pDocument;
}
#endif //_DEBUG
```

As you can see, the main purpose for both versions of GetDocument( ) is to return a pointer to the document itself. This may appear to be a very convoluted way of doing things, but it works.

Make absolutely certain that you place an #include directive for the document class file at the beginning of the HelpView.H file like this: #include "HelpDoc.h." Otherwise, you'll get a lot of errors when you attempt to compile the code.

Now that we have access to the document, we can begin to do something with it. The first thing we'll need to do is add an OnInitialUpdate( ) method. This method determines what happens when the view gets created. One of the more important tasks that get performed by this method is navigating the HTML view to the correct file. The default code is set up to work with Web sites and direct location entries. We need to change it to work with an entry in the Open File dialog.

Listing 7-6 shows the code that you'll need to add. (Make certain that you change the hard-coded URL to a Web site that you can access from your test machine — the example shows an URL for the author's test server.)

Listing 7-6: **OnInitialUpdate( ) Method Source Code**

```
void CHelpView::OnInitialUpdate()
{
    CHelpDoc*    pDoc = GetDocument();    // Document pointer

    // Determine if the user has selected a document. If they
    // have, then go to that document. Otherwise, display the
    // help desk Web site.
    if (!pDoc->GetPathName().IsEmpty())
        Navigate2(_T(pDoc->GetPathName()));
    else
        Navigate2(_T("http://nt_server"));
}
```

As you can see, we create a pointer to the document object. Once the document object is available, the program can determine whether it needs to process a file from the local hard drive, or give the user access to the help desk located on a Web server. The Navigate2( ) function accepts the name of the file or a URL as the first argument. (There are other arguments for this function that we won't discuss fully in this section that allow you to choose a specific frame on the Web page and set flags that determine how the call interacts with Internet Explorer features such as the history list.)

Adding the Movement Handlers

Just displaying the Web page probably won't do much for the user. They'll also need the capability of moving from one page to the next using the Forward and Backward buttons that we defined earlier. The next thing we'll need to do then is create two movement handlers, one for Forward and the other for Backward. Listings 7-7 and 7-8 show the code you'll need to add to the movement handlers once you create them.

Listing 7-7: **OnGoForward( ) Movement Handler Source Code**

```
void CHelpView::OnGoForward()
{
    // Go forward in the history list.
    GoForward();

}
```

Listing 7-8: **OnGoBack( ) Movement Handler Source Code**

```
void CHelpView::OnGoBack()
{
    // Go Backward in the history list.
    GoBack();

}
```

As you can see, the code for adding the movement required on a Web page is amazingly easy because it's already implemented as part of the CHtmlView class. (Actually, the class only acts as a wrapper for the functionality built into Internet Explorer.) All you really need to do is call the built-in methods as required to get to where you want to go. It's this built-in functionality that makes using the CHtmlView class so easy to use.

Changing the Toolbar

We defined a special HTML view toolbar earlier in the chapter. Unfortunately, the toolbar won't be used automatically as part of the template; you have to implement this behavior independently. In this section, we'll look at one of the easier techniques you can use to provide a separate toolbar for each view that your application supports.

The first thing we'll need to do is define two new functions in the CMainFrame class. The first function will load the IDR_MAINFRAME toolbar, whereas the second will load the special IDR_HTMLVIEWTYPE toolbar. You could probably use just one function to do this, but using separate functions has the advantage of making the routines self-documenting and provides a very slight performance enhancement. Listing 7-9 shows the method declaration code you'll need to add to the MainFrm.H file and Listing 7-10 shows the actual method code you'll need to add to the MainFrm.CPP file.

Listing 7-9: **Toolbar-changing Method Declarations**

```
// Operations
public:
    // Functions required to change the toolbar
    // between views.
    void UseHelpToolbar();
    void UseMainToolbar();
```

Listing 7-10: **Toolbar-changing Method Source Code**

```
void CMainFrame::UseHelpToolbar()
{
    // Load the HTML view toolbar.
    m_wndToolBar.LoadToolBar(IDR_HELPVIEWTYPE);

    // Display the change.
    m_wndToolBar.RedrawWindow();
}

void CMainFrame::UseMainToolbar()
{
    // Load the main toolbar.
    m_wndToolBar.LoadToolBar(IDR_MAINFRAME);

    // Display the change.
    m_wndToolBar.RedrawWindow();
}
```

You must perform two steps when switching toolbars. The first is to load the toolbar into the existing CToolBar object. This object always appears in the CMainFrame class of your application. In this case, the name of the variable is m_wndToolBar, which is protected. Because this variable is protected, you must make any changes to the toolbar from within the CMainFrame class. Once you have the new toolbar loaded, you display it using the RedrawWindow( ) method. In most cases, the default settings for RedrawWindow( ) will produce good results.

Now that we have the methods required to change the toolbars in place, we need some code to call these methods when a particular view window gets the focus. That means adding a new method to both the text view and the HTML view. Both of these functions work the same way; the biggest difference is that each view has to load the toolbar intended for its use. Listing 7-11 shows the new function you'll need to create for the HTML view, whereas Listing 7-12 shows the new function for the text view.

Listing 7-11: **HTML View OnSetFocus( ) Method Source Code**

```
void CHelpView::OnSetFocus(CWnd* pOldWnd)
{
    CMainFrame*    pFrame;    // The main frame pointer.

    // Peform the default action.
    CHtmlView::OnSetFocus(pOldWnd);

    // Obtain a pointer to the main frame.
    pFrame = (CMainFrame*)pOldWnd->GetParentFrame();

    // Tell the main frame to use the HTML view toolbar.
    pFrame->UseHelpToolbar();
}
```

Listing 7-12: **Text View OnSetFocus( ) Method Source Code**

```
void CHTMLViewView::OnSetFocus(CWnd* pOldWnd)
{
    CMainFrame*    pFrame;    // A pointer to the main frame.

    // Perform the default action.
    CEditView::OnSetFocus(pOldWnd);

    // Obtain a pointer to the main frame.
```

Continued

Listing 7-12: *(continued)*

```
    pFrame = (CMainFrame*)pOldWnd->GetParentFrame();

    // Tell the main frame to use the HTML view toolbar.
    pFrame->UseMainToolbar();

}
```

These two functions work essentially the same, so I'll discuss them both at the same time. The first task the code needs to perform is to call the superclass method so that the default processing takes place. Once the default processing is finished, the function can create a pointer to the CMainFrame class. Remember that you can't access the toolbar member variable directly. So the next step is to call the function we created to load a new toolbar from within the CMainFrame class itself.

Defining a Help Link

At this point, you could compile the application and it would be able to open both HTML and text files. In addition, the toolbar would automatically change to meet the needs of whichever view you're using. It's finally time to create the help link.

Four steps are required to create the help link (this section will help you do everything but step 4, which is an individual task for your particular company).

1. Determine which views will gain access to the help link and if you want help available if there are no views open (most applications use a separate menu when no views are open, so this is an important consideration).
2. Create the required menu entries in each view that will have a help link.
3. Add code to each view that handles the help-link menu entry command.
4. Define the help-link Web site.

Defining the required menu entries and associated event handler is relatively easy — the same set of steps will work for each view. The following procedure will show you how.

1. Open the menu for the desired view. For example, if you wanted to add a help link for the HTML view, you'd open the IDR_HELPVIEWTYPE menu.
2. Double-click the first blank entry in the Help submenu.
3. Type **ID_HELP_DESK** in the ID field, **&Help Desk** in the Caption field, and **Allows you to access the help desk.\nHelp Desk** in the Prompt field.

4. Close the menu.
5. Ctrl+Double-click the new Help Desk menu entry. You'll see the MFC ClassWizard dialog shown in Figure 7-18. This is where you'll add the COMMAND event handler. Visual C++ should select all of the required entries for you automatically, as shown in Figure 7-18. If it doesn't, you'll need to choose the correct object manually.

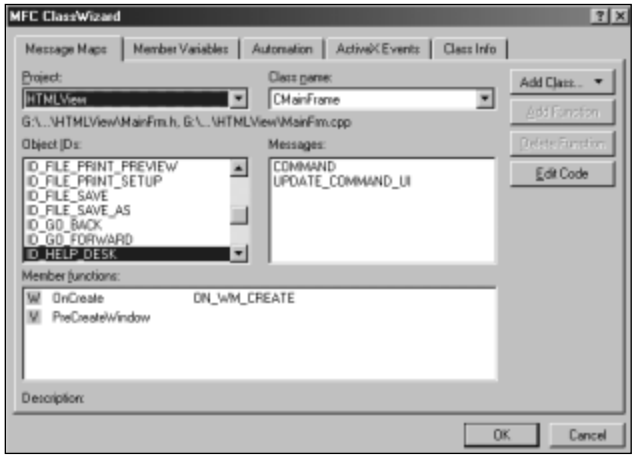


Figure 7-18: The MFC ClassWizard will allow you to create a message handler for the new Help Desk menu entry.

6. Highlight the COMMAND message, then click Add Function. You'll see an Add Member Function dialog like the one shown in Figure 7-19.

Figure 7-19: The Add Member Function enables you to create a new message handler.

7. Click OK to add the new message handler.
8. Click Edit Code to go to the new message handler.
9. Repeat steps 1 through 8 for all of the views that you want to add a help link to.

Unlike other kinds of Web page movement we've discussed in this chapter, the handling of a help link within a view differs by the kind of view that you're working with. In this case, there is one method for views based on CHtmlView and another method for views that are based on other classes. Listing 7-13 shows the code you'll need to add to handle a help link with a view based on CHtmlView.

Listing 7-13: CHtmlView Class-based View Help-desk Link Source Code

```
void CHelpView::OnHelpDesk()
{
    // Go to the help desk page.
    Navigate2(_T("http://nt_server"), NULL, NULL);

}
```

As you can see, the built-in CHtmlView class functionality comes into play with this view again. All we need to do is use the Navigate2( ) method to go to the help desk, or any other HTML Web site.

Working with a view based on something other than CHtmlView requires a lot of additional work because the view is incapable of displaying a Web page. This means that we first have to create a view capable of displaying the Web page, then load the desired site manually. Listing 7-14 shows the code that we'll need to use for any non-ChtmlView-based view (including instances in which there is no view loaded at all).

Listing 7-14: Other Class-based View Help-desk Link Source Code

```
void CHTMLViewView::OnHelpDesk()
{

  // Create a template for displaying the HTML view.
  CMultiDocTemplate*  pTemplate;

  pTemplate = new CMultiDocTemplate(
    IDR_HELPVIEWTYPE,
    RUNTIME_CLASS(CHelpDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CHelpView));
```

```
  // Use the template's OpenDocumentFile() method to create a blank
  // view.  Save the CDocument object that this call returns as a
  // CHelpDoc class object.
  CHelpDoc*  pDoc = (CHelpDoc*)pTemplate->OpenDocumentFile(NULL, TRUE);

  // Change the document title from "Untitled" to "Help Desk".
  pDoc->SetTitle("Help Desk");
}
```

This is a lot of work compared to the code for the ChtmlView-based view. The first thing we need to do is create a template. This template code should look familiar since we added it earlier to the CMainFrame class to allow the application to open more than one type of document. Notice that the resource ID, document class, and view class are all unique to the HTML view that we created earlier.

Once the template is created, the code can use it to create a blank view. The application will use the template to create everything needed, including a frame. Notice that we haven't specified the name of a document to use. Asking the CHelpView class to create a new view without supplying a class name means that it'll automatically open the Help Desk Web page. Now you can see why the additional code in Listing 7-6 was required. The else part of the if statement in that listing tells CHelpView to automatically navigate to the Help Desk if the path name is blank.

The only thing left to do is assign a title to the new view. Otherwise, your application will display a name of Untitled rather than something descriptive like Help Desk. At this point, you can compile the code, and the help link is fully functional.

Working with Internet Explorer 5

This chapter has covered a specialty browser application type that enables you to create a custom approach to Internet use. However, you may decide that you don't want to limit your employee's access to the Internet or that you don't want to create a custom browser. In addition, it may not be feasible to create a custom setup if you also have to support access to the Web site by partner companies or consultants. Internet Explorer 5 provides a vast amount of capabilities that you can tap to create a custom look for your Web site using a standard browser; all you need to do is create a Web site that uses a combination of client- and server-side processing.

Continued

(continued)

The advantage of using Internet Explorer is that everyone will be able to access your Web site equally. The disadvantage is that you've opened the Internet for complete use by everyone. In other words, although you do gain flexibility and accessibility, you lose control over where the employee can go.

Internet Explorer 5 also has new features not found in older versions of the product that you would have a difficult time adding to a custom Web browser. The most important of these features is IntelliSense, which is a collection of Microsoft technologies designed to automate common tasks, or at least make them easier to perform. IntelliSense makes it easier to search for information on the Internet as a whole. It also makes the process of integrating portals, search engines, and applications with the browser, much easier. In sum, IntelliSense is an essential new feature if you want to get the maximum flexibility from your browser and if you're a Microsoft shop. As of this writing, you can download the latest version of Internet Explorer 5 at `http://www.microsoft.com/windows/ie/ie5/default.asp`.

Performance is also a big issue for Internet Explorer 5. One of the new enhancements deals with HTML expires. In the past, Internet Explorer used to guess whether data stored in the cache was actually stale. In many cases, this resulted in unnecessary network traffic, which slowed performance both locally and at the network level. Internet Explorer 5 makes better use of HTML expires and reads data from the network only when the cached data has actually expired. This means that, in some cases, you'll get better performance from Internet Explorer than what you'll get with a custom application.

Some DHTML enhancements are also provided with Internet Explorer 5 that will be available to your custom application once you have the new browser installed. These enhancements include support for CSS Level 2, dynamic properties, and dynamic HTML behavior. CSS Level 2 has been approved by W3C, but the other two additions have not (they have been submitted to W3C for approval as of this writing).

Besides feature additions, Internet Explorer 5 boasts two speed enhancements for DTHML. First, Microsoft has optimized the algorithms used to read and execute DHTML script. These optimizations are most apparent when working with large or data-bound Web pages. Second, the Web-page author can now include hints that tell Internet Explorer 5 what to expect from the Web page. Depending on how well the hints match what the browser actually finds on the Web page, you could see a significant speed increase for any size Web page.

Summary

The CHtmlView class provides many opportunities for a developer to make use of the Internet (or a company intranet) as a new means for distributing information.

Whether that distribution is a help desk or simply provides the means to better monitor employee activity, custom browser applications can make a lot of difference in how you deal with the Internet as a whole. We didn't fully cover every potential use of the CHtmlView class in this chapter, which means you shouldn't limit yourself to the ideas presented here. For example, some people actually use the CHtmlView class to create custom programming utilities — an idea that we didn't cover at all. Consider this chapter to be a starting point to an entirely new world of application programming ideas. Here are some additional conclusions you can draw from the contents of this chapter:

- Uses for Web pages are no longer relegated to the Internet, you can use them for other purposes like creating very flexible help systems.
- The CHtmlView class allows you to create a custom Web browser using Visual C++.
- Even though there is only one Web browser application type, there are two main ways to use it: client-side or server-side processing.
- Client-side processing allows you to gain the greatest benefit from a custom Web browser, including server load reduction, processing speed, reduced network traffic, customization, lower training and support costs, and special features.
- DHTML enables you to enhance the functionality of Web pages through the use of scriptlets and formatting features.
- All DHTML scriptlets have access to an event object with clearly defined properties.
- Not all event object properties will contain useful information even though all event object properties are available all of the time.
- XML is a future addition to the tools at the programmer's disposal that will augment, not replace, DHTML.
- Many of the new Internet controls provided with Visual C++ serve more than one purpose in both Internet and desktop-related applications.
- Adding a view to an application means adding both the required view and document classes to the project. In addition, you need to add any required resources, document templates, and message handlers.
- The view class determines how the data gets presented onscreen, whereas the document class determines how the data is manipulated by the user.
- A document template will control most customized aspects of a view except for any special toolbars that your application may require.
- When working with some types of views, you'll need to gain access to the document object in order to track user actions like opening or saving files.

✦ ✦ ✦